I F I G
Research
Report

Institut für Informatik

Fault Tolerant Parallel Pattern Recognition

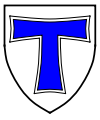
Martin Kutrib, Jan-Thomas Löwe

IFIG Research Report 0001

February 2000

Institut für Informatik
JLU Gießen
Arndtstraße 2
D-35392 Giessen, Germany
Tel: +49-641-99-32141
Fax: +49-641-99-32149
mail@informatik.uni-giessen.de
www.informatik.uni-giessen.de

Justus-Liebig-



Universität
Giessen

IFIG Research Report
IFIG Research Report 0001, February 2000

Fault Tolerant Parallel Pattern Recognition

Martin Kutrib[1], Jan-Thomas Löwe[2]

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany

Abstract. The general capabilities of fault tolerant computations in one-way and two-way linear cellular arrays are investigated in terms of pattern recognition.
The defective processing elements (cells) that cause the misoperations are assumed to behave as follows. Dependent on the result of a self-diagnosis they store their working state locally such that it becomes visible to the neighbors. A non-working (defective) cell cannot modify information but is able to transmit it unchanged with unit speed. Arrays with static defects run the self-diagnosis once before the actual computation. Subsequently no more defects may occur. In case of dynamic defects cells may fail during the computation.
We center our attention to patterns that are recognizable very fast, i.e. in real-time, but almost all results can be generalized to arbitrary recognition times in a straightforward manner. It is shown that fault tolerant recognition capabilities of two-way arrays with static defects are characterizable by intact one-way arrays and that one-way arrays are fault tolerant per se.
For arrays with dynamic defects it is proved that the failures can be compensated as long as the number of adjacent defective cells is bounded. Arbitrary large defective regions (and thus fault tolerant computations) lead to a dramatically decrease of computing power. The recognizable patterns are those of a single processing element, the regular ones.

CR Subject Classification (1998): F.1, F.4.3, B.6.1, E.1, B.8.1, C.4

---

[1]E-mail: kutrib@informatik.uni-giessen.de
[2]E-mail: j.loewe@informatik.uni-giessen.de

1 Introduction

Nowadays it becomes possible to build massively parallel computing systems that consist of hundred thousands of processing elements. Each single component is subject to failure such that the probability of misoperations and loss of function of the whole system increases with the number of its elements. It was von Neumann [10] who first stated the problem of building reliable systems out of unreliable components. Biological systems may serve as good examples. Due to the necessity to function normally even in case of certain failures of their components the nature developed mechanisms which invalids the errors, they are working in some sense fault tolerant. Error detecting and correcting components should not be global to the whole system because they themselves are subject to failure. Therefore the fault tolerance has to be a design feature of the single elements.

A model for massively parallel, homogenously structured computers are the cellular arrays. Such devices of interconnected parallel acting finite state machines have been studied from various points of view. Fault tolerant computations have been investigated, e.g. in [1, 7] where encodings are established that allow the correction of so-called K-separated misoperations, in [3, 4, 9, 11] where the famous firing squad synchronization problem is considered in defective cellular arrays, and in terms of interacting automata with nonuniform delay in [2, 5] where the synchronization of the networks is the main object also.

Here we are interested in more general computations. In terms of pattern recognition the general capabilities of fault tolerant computations are considered. Since cellular arrays have intensively been investigated from a language theoretic point of view, pattern recognition (or language acceptance) establishes the connection to the known results and, thus, inheres the possibility to compare the fault tolerant capabilities to the non fault tolerant ones.

In the sequel we distinguish two different types of defects.

Static defects are the main object of Section 3. It is assumed that each cell has a self-diagnosis circuit which is run once before the actual computation. The results are stored locally in the cells and subsequently no new defects may occur. Otherwise the whole computation would become invalid. A defective cell cannot modify information but is able to transmit it with unit speed. Otherwise the parallel computation would be broken into two non interacting parts and, therefore, would become impossible at all.

In section 4 the defects are generalized. In cellular arrays with dynamic defects it may happen that a cell becomes defective at any time. The formalization of the corresponding arrays includes also the possibility to repair a cell dynamically.

In the following section we define the basic notions and recall the underlying intact cellular arrays and their mode of pattern recognition.

2 Basic notions

We denote the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \cdots\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. $X_1 \times \cdots \times X_d$ denotes the Cartesian product of the sets $X_1, \ldots, X_d$. If $X_1 = \cdots = X_d$ we use the notion X_1^d alternatively. We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. Let M be some set and $f : M \to M$ be a function, then we denote the i-fold composition of f by $f^{[i]}$, $i \in \mathbb{N}$.

A two-way resp. one-way cellular array is a linear array of identical finite state machines, sometimes called cells, which are connected to their both nearest neighbors resp. to their nearest neighbor to the right. The array is bounded by cells in a distinguished so-called boundary state. For convenience we identify the cells by positive integers. The state transition depends on the current state of each cell and the current state(s) of its neighbor(s). The transition function is applied to all cells synchronously at discrete time steps. Formally:

Definition 1 *A* two-way cellular array *(*CA*) is a system* $\langle S, \delta, \#, A \rangle$*, where*

1. *S is the finite, nonempty set of* cell states,
2. *$\# \notin S$ is the* boundary state,
3. *$A \subseteq S$ is the set of* input symbols,
4. *$\delta : (S \cup \{\#\})^3 \to S$ is the* local transition function.

If the flow of information is restricted to one-way (i.e. from right to left) the resulting device is a *one-way cellular array* (OCA) and the local transition function maps from $(S \cup \{\#\})^2$ to S.

A *configuration* of a cellular array at some time $t \geq 0$ is a description of its global state, which is actually a mapping $c_t : [1, \ldots, n] \to S$ for $n \in \mathbb{N}$.

The data on which the cellular arrays operate are patterns built from input symbols. Since here we are studying one-dimensional arrays only the input data are finite strings (or words). The set of strings of length n built from symbols from a set A is denoted by A^n, the set of all such finite strings by A^*. We denote the *empty string* by ε and the *reversal of a string* w by w^R. For its length we write $|w|$. A^+ is defined to be $A^* \setminus \{\varepsilon\}$.

In the sequel we are interested in the subsets of strings that are recognizable by cellular arrays. In order to establish the connection to formal language theory we call such a subset a *formal language*. Moreover, sets L and L' are considered to be equal if they differ at most by the empty word, i.e. $L \setminus \{\varepsilon\} = L' \setminus \{\varepsilon\}$.

Now we are prepared to describe the computations of (O)CAs. The operation starts in the so-called *initial configuration* $c_{0,w}$ at time 0 where one symbol of the input string $w = x_1 \cdots x_n$ is fed to one cell, respectively: $c_{0,w}(i) = x_i$, $1 \leq i \leq n$. During a computation the (O)CA steps through a sequence of configurations whereby successor configurations are computed according to the global transition function Δ: Let c_t, $t \geq 0$, be a configuration, then its successor

configuration is as follows:

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(1) = \delta\big(\#, c_t(1), c_t(2)\big) \\ & c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \in \{2, \ldots, n-1\} \\ & c_{t+1}(n) = \delta\big(c_t(n-1), c_t(n), \#\big) \end{aligned}$$

for CAs and

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(i) = \delta\big(c_t(i), c_t(i+1)\big), i \in \{1, \ldots, n-1\} \\ & c_{t+1}(n) = \delta\big(c_t(n), \#\big) \end{aligned}$$

for OCAs. Thus, Δ is induced by δ.

An input string w is recognized by an (O)CA if at some time i during its course of computation the leftmost cell enters a final state from the *set of final states* $F \subseteq S$.

Definition 2 *Let $\mathcal{M} = \langle S, \delta, \#, A\rangle$ be an* (O)CA *and $F \subseteq S$ be a set of final states.*

1. *An input $w \in A^*$ is* recognized *by $\mathcal{M}$ if it is the empty string or if there exists a time step $i \in \mathbb{N}$ such that $c_i(1) \in F$ holds for the configuration $c_i = \Delta^{[i]}(c_{0,w})$.*
2. *$L(\mathcal{M}) = \{w \in A^* \mid w$ is recognized by $\mathcal{M}\}$ is the* set of strings (language) recognized *by $\mathcal{M}$.*
3. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping and i_w be the minimal time step at which $\mathcal{M}$ recognizes $w \in L(\mathcal{M})$. If all $w \in L(\mathcal{M})$ are recognized within $i_w \leq t(|w|)$ time steps, then L is said to be of* time complexity *t.*

The family of all sets which are recognizable by some CA (OCA) with time complexity t is denoted by $\mathscr{L}_t(\text{CA})$ ($\mathscr{L}_t(\text{OCA})$). If t equals the identity function $id(n) = n$ recognition is said to be in *real-time*, and if t is equal to $k \cdot id$ for an arbitrary rational number $k \geq 1$ then recognition is carried out in *linear-time*. Correspondingly, we write $\mathscr{L}_{rt}((\text{O})\text{CA})$ and $\mathscr{L}_{lt}((\text{O})\text{CA})$. In the sequel we will use corresponding notations for other types of recognizers.

3 Static defects

Now we are going to explore some general recognition capabilities of CAs that contain some defective cells. The defects are in some sense static [9]: It is assumed that each cell has a self-diagnosis circuit which is run before the actual computation. The result of that diagnosis is stored in a special register of each cell such that intact cells can detect defective neighbors. Moreover (and this is the static part), it is assumed that during the actual computation no new defects may occur. Otherwise the whole computation would become invalid. What is the effect of a defective cell? It is reasonable to require that a defective

cell cannot modify information. On the other hand, it must be able to transmit information in order to avoid the parallel computation being broken into two not interacting lines and, thus, being impossible at all.

The speed of information transmission is one cell per time step. Another point of view on such devices is to define a transmission delay between every two adjacent cells and to allow nonuniform delays [2, 5]. Now the number of defective cells between two intact ones determine the corresponding delay.

Since the self-diagnosis is run before the actual computation we may assume that defective cells do not fetch an input symbol. Nevertheless, real-time is the minimal possible time needed for non-trivial computations and, consequently, is defined to be the number of all cells in the array. In order to obtain a computation result here we require the leftmost cell to be not defective. Later on we can omit this assumption.

Formally we denote CAs with static defects by SD-CA and the corresponding language families by $\mathscr{L}_t(\text{SD-CA})$.

Considering the general real-time recognition capabilities of SD-CAs the best case is trivial. It occurs when all the cells are intact: The capabilities are those of CAs. On the other hand, fault tolerant computations are concerned with the worst case (with respect to our assumptions on the model). The next two results show that in such cases the capabilities can be characterized by intact OCAs from what follows that the bidirectionality of the information flow gets lost.

Theorem 3 *If a set is fault tolerant real-time recognizable by a* SD-CA *then it is real-time recognizable by an* OCA.

Proof. Let $\mathcal{D}$ be a SD-CA and let $k \in \mathbb{N}$ be an arbitrary positive integer. Set the number of cells of $\mathcal{D}$ to $n = 2^k - 1$. For the mapping $f : \mathbb{Z} \to \mathbb{Z}$, $f(z) = n - 2^z + 2$ holds: $\forall\, z \in \{1, \ldots, k\} : f(z) \in \{1, \ldots, n\}$.

Now assume the cells at the positions $f(i)$, $1 \leq i \leq k$, are intact ones and all the other cells are defective (see Figure 1). In between the cells $f(i)$ and $f(i+1)$, $1 \leq i \leq k-1$, there are $f(i) - f(i+1) - 1 = (n - 2^i + 2) - (n - 2^{i+1} + 2) - 1 = 2^{i+1} - 2^i - 1 = 2^i - 1$ defective ones.

During a real-time computation the states of a cell $f(i)$ at time $t \geq 2^i$ cannot influence the overall computation result (see Figure 1). The states would reach the leftmost cell after another $f(i)-1 = (n-2^i+2)-1 = 2^k-1-2^i+1 = 2^k-2^i$ time steps. This gives the arrival time $2^i+2^k-2^i = 2^k = n+1$, which is greater than real-time.

Conversely, the cell $f(i)$ computes all its states up to time $t \leq 2^i - 1$ independently on the states of its intact neighbors to the left: The nearest intact neighbor to the left is cell $f(i+1)$ and there are $2^i - 1$ defective cells in between $f(i+1)$ and $f(i)$.

Up to now we have shown that the information flow in $\mathcal{D}$ is one-way. But compared to OCAs the cells in $\mathcal{D}$ are performing more state changes. It remains to show that this does not lead to stronger capabilities.

Let i be some intact cell of $\mathcal{D}$. As long as it operates independently on its intact neighbors it runs through state cycles provided that the adjacent defective regions are long enough. Let $s_0 s_1 \cdots s_j s_{j+1} \cdots s_{j+k} s_j \cdots$ be such a cycle. Now one can always enlarge the lengths of the defective regions such that they correspond to $j + p \cdot (k+1)$, $p \in \mathbb{N}_0$.

Therefore, during their isolated computations the cells run through complete cycles. Obviously, such a behavior can be simulated by the cells of an OCA since the cycle lengths are bounded by the number of states of $\mathcal{D}$. □

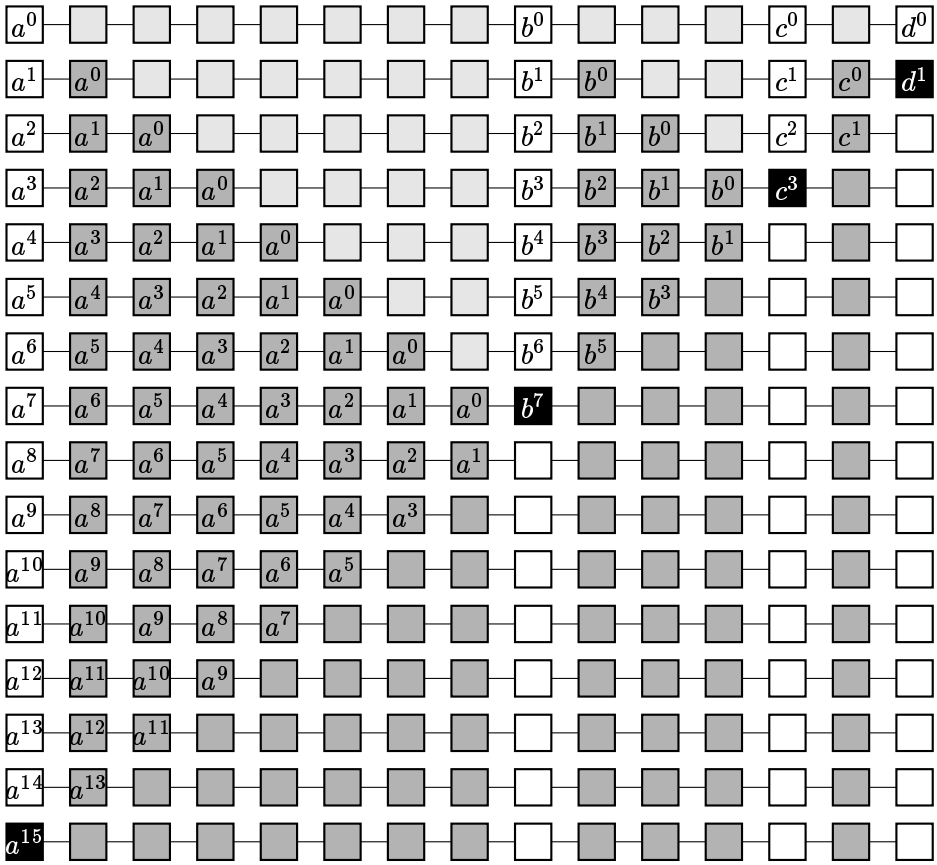


Figure 1: One-way information flow in SD-CAs.

In order to obtain the characterization of real-time SD-CAs by real-time OCAs we need the converse of Theorem 3.

Theorem 4 *If a set is real-time recognizable by an* OCA *then it is fault tolerant real-time recognizable by a* SD-CA.

Proof. The idea of the simulation is depicted in Figure 2. Each cell of a SD-CA that simulates a given OCA waits for the first information from its right intact neighbor. The waiting period is signaled to its left intact neighbor by signals labeled $*$. This information leads to a waiting period of the left

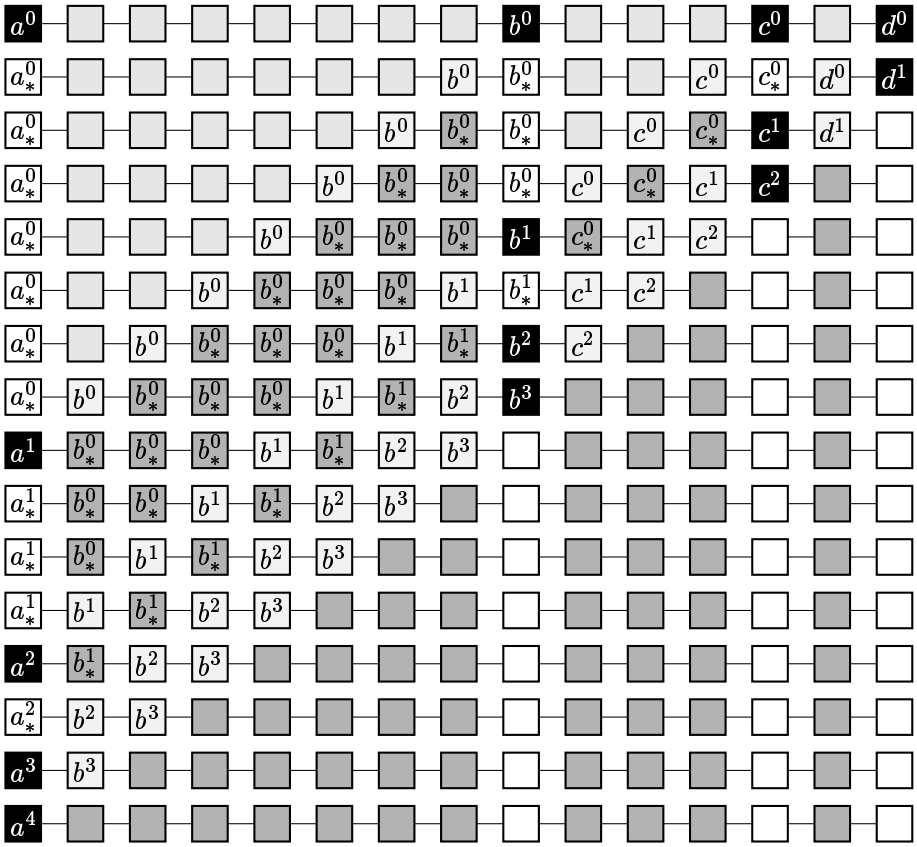


Figure 2: OCA simulation by SD-CAs.

intact neighbor. Each intact cell performs a simulation step when it receives a non-waiting signal.

It follows that a cell sends exactly as many waiting signals to the left as are defective cells located to its right. Therefore, the leftmost cell needs exactly one simulation step for each intact cell and one waiting step for each defective cell and, thus, computes the result in real-time. □

The following corollary formalizes the characterization:

Corollary 5 $\mathscr{L}_{rt}(\text{SD-CA}) = \mathscr{L}_{rt}(\text{OCA})$

From the previous results follows the interesting fact that OCAs are per se fault tolerant. Additional defective cells do not decrease the recognition capabilities.

Corollary 6 $\mathscr{L}_{rt}(\text{SD-OCA}) = \mathscr{L}_{rt}(\text{OCA})$

It is often useful to have examples for string sets not recognizable by a certain device.

Example 7 Neither the set of duplicated strings $\{ww \mid w \in A^*\}$ nor the set of strings whose lengths are a power of 2 $\{w \mid w \in A^* \text{ and } |w| = 2^i, i \in \mathbb{N}\}$ are fault tolerant real-time recognizable by SD-CAs.

(It has been shown in [6] resp. [8] that they do not belong to the family $\mathscr{L}_{rt}$(OCA).)

The previous results imply another natural question. Is it possible to regain the recognition power of two-way CAs in fault tolerant SD-CA computations by increasing the computation time? How much additional time would be necessary?

Without proof we present a piece of good news. Only one additional time step for each intact cell is necessary in order to regain the computation power in a fault tolerant manner.

Theorem 8 *If a set is real-time recognizable by a* CA *then it is fault tolerant recognizable by a* SD-CA *in time real-time+m, where m denotes the number of intact cells.*

One assumption on our model has been an intact leftmost cell. Due to Corollary 5 we can omit this requirement. Now the overall computation result is indicated by the leftmost intact cell of the one-way array which operates per se independently on its defective left neighbors.

4 Dynamic defects

In the following cellular arrays with dynamic defects (DD-CA) are introduced. Dynamic defects can be seen as generalization of static defects. Now it becomes possible that cells fail at any time during the computation. Afterwards they behave as in the case of static defects.

In order to define DD-CAs more formally it is helpful to suppose that the state of a defective cell is a pair of states of an intact one. One component represents the information that is transmitted to the left and the other one the information that is transmitted to the right. By this formalization we obtain the type indication of the cells (defective or not) for free: Defective cells are always in states from S^2 and intact ones in states from S. A possible failure implies a weak kind of nondeterminism for the local transition function.

Definition 9 *A two-way cellular array with dynamic defects (*DD-CA*) is a system $\langle S, \delta, \#, A\rangle$, where*

1. *S is the finite, nonempty set of* cell states *which satisfies $S \cap S^2 = \emptyset$,*
2. *$\# \notin S$ is the* boundary state,
3. *$A \subseteq S$ is the set of* input symbols,
4. *$\delta : (S \cup \{\#\} \cup S^2)^3 \to \{\{a, (b, c)\} \mid a, b, c \in S\}$ is the* local transition function *which satisfies*
   $\delta(s_1, s_2, s_3) = \{s, (s_l, s_r)\}$ with $s \in S$, $(s_1 = s_l \in S \vee s_1 = (s_l, s'_r) \in S^2), (s_3 = s_r \in S \vee s_3 = (s'_l, s_r) \in S^2)$

If a cell works fine the local transition function maps to a state from S. Otherwise it maps to a pair from S^2 indicating that the cell is now defective. The

definition includes the possibility to repair a cell during the computation. In this case δ would map from a pair to a state from S. Note that the nondeterminism in a real computation is a determinism since the failure or repair of a cell is in some sense under the control of the outside world.

We assume that initially all cells are intact and as in the static case that the leftmost cell remains intact.

In the sequel we call an adjacent subarray of defective cells a *defective region.*

The next results show that dynamic defects can be compensated as long as the lengths of defective regions are bounded.

Theorem 10 *If a set is real-time recognizable by a* CA *then it its real-time recognizable by a* DD-CA *if the lengths of its defective regions are bounded by some* $k \in \mathbb{N}_0$.

Proof. Assume for a moment that the lengths of the defective regions are exactly k. A DD-CA $\mathcal{D}$ that simulates a given CA $\langle S, \delta, \#, A\rangle$ has the state set $S' = S^{4k+1}$.

The general idea of the proof is depicted in Figure 3. As long as a cell does not detect a defective neighbor it stores the states of its neighbors and its own state in some of its additional registers as shown in the figure.

At time t the state of cell i might be as follows:

$$(\ldots, c_{t-1}(i-1), c_{t-1}(i), \underbrace{c_t(i)}_{\text{center}}, c_{t-1}(i), c_{t-1}(i+1), \ldots)$$

Assume now that the right neighbor of cell i becomes defective. Due to our assumption we know that there must exist a defective region of length k at the right of cell i. During the next k time steps cell i stores the received states and computes missing states from its register contents as shown in Figure 3.

Subsequently its state might be as follows

$$(\ldots, \underbrace{c_{t+k}(i)}_{\text{center}}, c_{t+k-1}(i), c_{t+k-1}(i+1), \ldots, c_{t+1}(i+k-2), c_t(i+k-1), c_t(i+k))$$

From now on cell i receives the states that the intact cell $i+k+1$ has been in at time $t, t+1, \ldots$ and is able to compute the necessary intermediate states from its register contents.

A crucial point is that the lenghts of defective regions are fixed to k. Due to that assumption a cell i knows when it receives the valid states from its next intact neighbor $i+k+1$ or $i-k-1$. We can relax the assumption as required to lengths of at most k cells by the following extension of the simulation. Each cell is equipped with a modulo k counter. Since the current value of the counter is part of the cell state it is also part of the transmitted information. A cell that stores received information in its additional registers stores also the received counter value. Now it can decide whether it receives the valid state from its next intact neighbor by comparing the received counter value to the latest stored counter value. If they are equal then the received information is from a defective cell, otherwise it is valid and the cell uses no more additional registers.

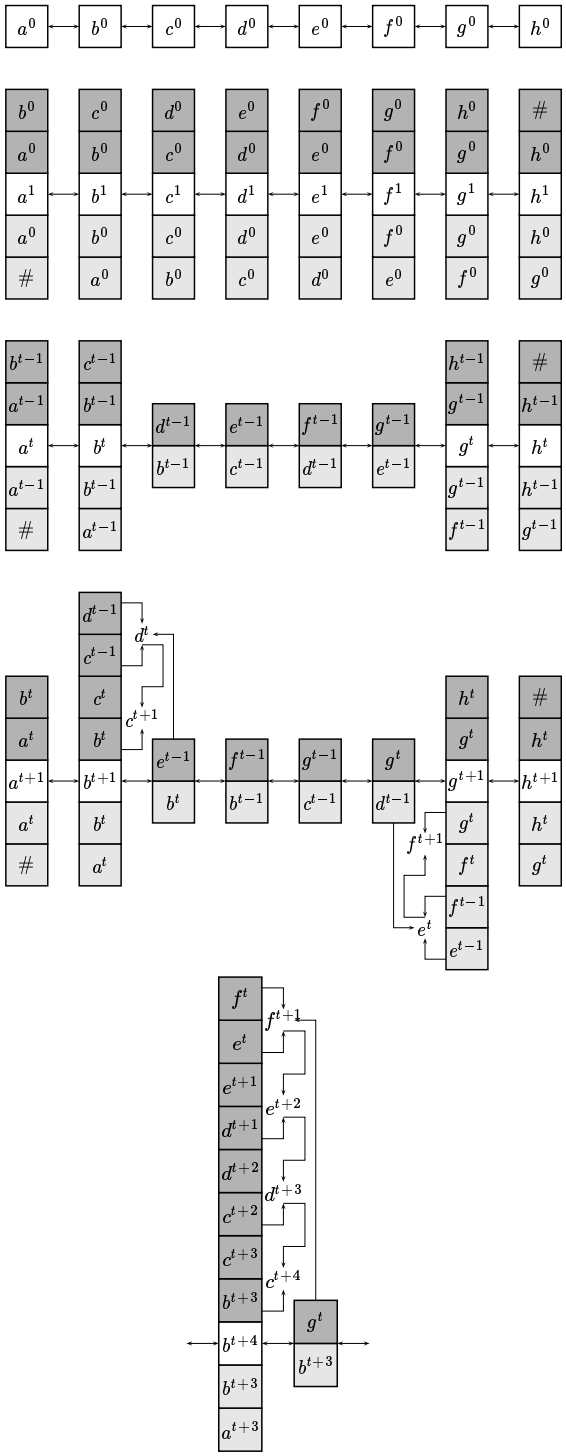


Figure 3: Compensation of $k = 4$ defects.

New failures in subsequent time steps can be detected by the same method. If the received counter value is equal to the latest stored counter value then additional cells have become defective. In such cases the cell uses correspondingly

more additional registers in order to compensate the new defects.

It remains to explain what happens if two defective regions are joint by failure of a single connecting intact cell. Up to now we have used the transmitted contents of the main registers only. But actually the whole state, i.e. all register contents, are transmitted. In the case in question the next intact cells to the left and right of the joint defective region can fill additional registers as desired. □

Corollary 11 *If a set is real-time recognizable by an* OCA *then it is real-time recognizable by a* DD-OCA *if the lengths of its defective regions are bounded by some* $k \in \mathbb{N}_0$.

In order to provide evidence for general fault tolerant DD-CA computations we have to relax the assumption of bounded defective region lengths. We are again concerned with the worst case. The hardest scenario is as follows. Initially all cells are intact and thus fetching an input symbol. During the first time step all but the leftmost cell fail. (Needless to say, if the leftmost cell becomes also defective then nobody would expect a reasonable computation result.)

It is easy to see that in such cases the recognition capabilities of DD-CAs are those of a single cell, a finite state machine (see Figure 4).

Lemma 12 *If a set is fault tolerant recognizable by a* DD-CA *then it is recognizable by a finite state machine and thus regular.*

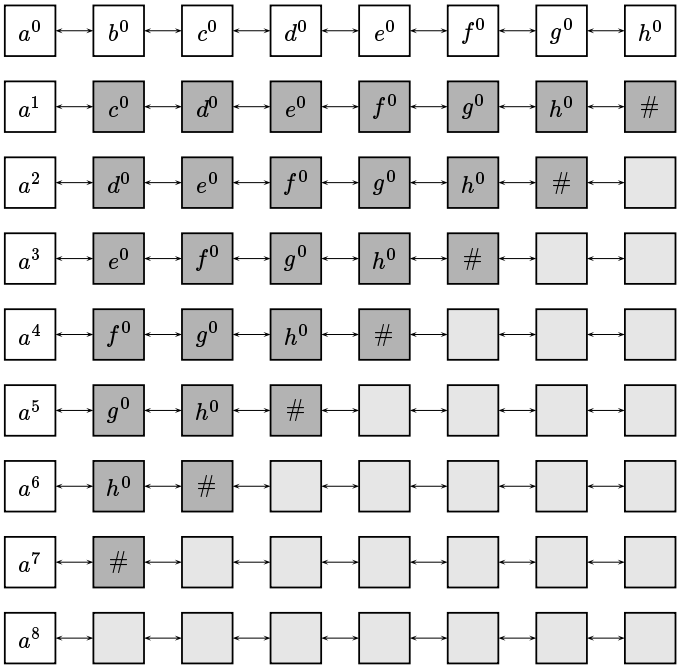


Figure 4: Worst case DD-CA computation.

Corollary 13 *If a set is fault tolerant recognizable by a* DD-OCA *then it is recognizable by a finite state machine and thus regular.*

References

[1] Harao, M. and Noguchi, S. *Fault tolerant cellular automata.* Journal of Computer and System Sciences 11 (1975), 171–185.

[2] Jiang, T. *The synchronization of nonuniform networks of finite automata.* Information and Computation 97 (1992), 234–261.

[3] Kutrib, M. and Vollmar, R. *Minimal time synchronization in restricted defective cellular automata.* Journal of Information Processing and Cybernetics EIK 27 (1991), 179–196.

[4] Kutrib, M. and Vollmar, R. *The firing squad synchronization problem in defective cellular automata.* IEICE Transactions on Information and Systems E78-D (1995), 895–900.

[5] Mazoyer, J. *Synchronization of a line of finite automata with nonuniform delays.* Research Report TR 94-49, Ecole Normale Supérieure de Lyon, Lyon, 1994.

[6] Nakamura, K. *Real-time language recognition by one-way and two-way cellular automata.* Mathematical Foundations of Computer Science 1999, LNCS 1672, 1999, 220–230.

[7] Nishio, H. and Kobuchi, Y. *Fault tolerant cellular spaces.* Journal of Computer and System Sciences 11 (1975), 150–170.

[8] Seidel, S. R. *Language recognition and the synchronization of cellular automata.* Technical Report 79-02, Department of Computer Science, University of Iowa, Iowa City, 1979.

[9] Umeo, H. *A fault-tolerant scheme for optimum-time firing squad synchronization.* Parallel Computing: Trends and Applications, 1994, 223–230.

[10] von Neumann, J. *Probabilistic logics and the synthesis of reliable organisms from unreliable components.* Automata Studies, Princeton University Press 34 (1956), 43–98.

[11] Yunès, J.-B. *Fault tolerant solutions to the firing squad synchronization problem.* Technical Report LITP 96/06, Institut Blaise Pascal, Paris, 1996.